Seminar

Software Engineering in sicherheitskritischen Systemen

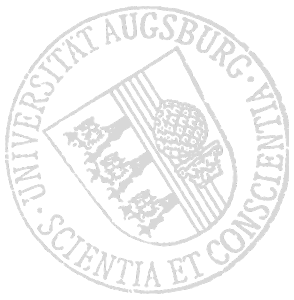
Sommersemester 2024

State of the Art: Compiler-based Static Code Analysis

Matrikelnummer:

Betreuer: Noël Hagemann
Softwaremethodik für verteilte Systeme (Prof. Bauer)
Universität Augsburg

Zusammenfassung Diese Seminararbeit behandelt das Thema der compilerbasierten statischen Codeanalyse. Dabei wird zuerst auf die Grundlagen des Softwaretestens und der statischen Codeanalyse eingegangen bevor dann an einem Beispiel der statischen Codeanalyse gefundene Fehler aufgezeigt werden. Wichtige Analysetechniken der Softwaretools, die zur Unterstützung bei der statischen Codeanalyse genutzt werden, sind die Datenflussanalyse und die Kontrollflussanalyse. Im Weiteren werden die drei Tools FindBugs, SonarQube und Snyk kurz vorgestellt. Zum Abschluss der Arbeit wird vor dem Schluss noch auf die Herausforderungen der statischen Codeanalyse eingegangen. Auch wenn der Aufwand groß ist, können mit der statischen Codeanalyse viele Fehler gefunden werden. Zur Unterstützung sind Tools eine wichtige Hilfe. Für größtmögliche Softwarequalität ist das Zusammenspiel mit weiteren Testverfahren unverzichtbar.


UNIVERSITÄT AUGSBURG · SCIENTIA ET CONSCIENTIA

Inhaltsverzeichnis

1 Einleitung

Unabhängig von der Funktionalität ist der Anspruch an eine Software, sicher und zuverlässig zu funktionieren. Die Codequalität spielt dabei eine wichtige Rolle. Statische Codeanalysewerkzeuge sind ein nützliches Hilfsmittel, um die Codequalität deutlich zu verbessern. Sie helfen potenzielle Fehler zu entdecken und zu beheben. Welche Fehler das sind und wie die statische Codeanalyse funktioniert wird in dieser Arbeit beschrieben.

Die Arbeit beginnt in Kapitel 2 mit Grundlagen des Softwaretestings und der statischen Codeanalyse. Kapitel 3 zeigt ein Beispiel für die auffindbaren Fehler durch die statische Codeanalyse und erklärt zwei wichtige Analysetechniken Datenflussanalyse und Kontrollflussanalyse. Verschiedene Analysetools, die zur Unterstützung genutzt werden, werden ebenfalls in Kapitel 3 vorgestellt. Im Anschluss wird in Kapitel 4 noch auf die Herausforderungen und Grenzen der statischen Codeanalyse eingegangen, bevor in Kapitel 5 der Zukunftsausblick und das Fazit folgen.

2 Softwaretesting

Im Bereich Softwaretesting werden verschiedene Verfahren genutzt. Im ersten Teil dieses Kapitels wird auf allgemeine Grundlagen eingegangen. Der zweite Teil beschreibt die statische Codeanalyse genauer.

2.1 Grundlagen des Softwaretestings

Ziel des Softwaretestings ist das möglichst vollständige Auffinden von Fehlern. Es lässt sich andersrum aber nicht zeigen, dass keine Fehler mehr im Programm vorhanden sind.

Sobald die zu testende Software komplexer wird, gibt es keine Möglichkeit mehr, die Software vollständig mit allen möglichen Kombinationen der Einstellungen und Eingabeparametern zu testen. Das Testen von Software ist immer nur ein Ausschnitt und wird je nach späterem Einsatz der Software intensiviert. Je größer das spätere Risiko und der Schaden bei einer Fehlfunktion wären, desto mehr Zeit und Geld wird in das Testen der Software investiert. [1, S. 37]

Es gibt zwei grundlegende Verfahren Software zu testen. Das sind die statische und die dynamische Analyse.

2.2 Statische Codeanalyse

Die statische Codeanalyse ist ein Whitebox Testverfahren. Das heißt, es wird ausschließlich der Quellcode zur Analyse benötigt und keine weitere Soft- oder Hardware. Diese

Analyse erfolgt ohne tatsächliche Ausführung des Codes, deshalb statische Codeanalyse. Sie hilft dabei, die Qualität, Sicherheit und Zuverlässigkeit von Softwareprojekten signifikant zu verbessern. Da der Quellcode alleine ausreicht und nicht ausgeführt werden muss, ist die statische Codeanalyse eine Möglichkeit, Fehler bereits in frühen Entwicklungsphasen zu finden bevor diese zu schwerwiegenden Problemen führen. Die statische Codeanalyse ist somit ein entscheidender Schritt, um die Effizienz und Robustheit von Softwareanwendungen zu gewährleisten.

Bei der statischen Codeanalyse wird die Einhaltung von Regeln überprüft, die individuell nach Bedarf festgelegt werden. Es können dabei alle syntaktischen Fehler gefunden werden, also beispielsweise nicht initialisierte Variablen, implizierte Datenteypkonvertierungen, schlechte Variablenbenennungen, inkorrekte Exceptions, mögliche Speicherüberläufe und viele weitere. Es kann aber auch auf Programmierstandards überprüft werden, sowohl firmeninterne als auch offizielle Programmierguidelines wie MISRA C Standards. Logische Fehler, die korrekte Funktionalität der Software oder einzelner Teile davon, sowie das Zusammenspiel mit weiterer Soft- oder Hardware können mit der statischen Codeanalyse nicht sichergestellt werden. Dazu werden dynamische Tests benötigt. Die statische Codeanalyse kann per Hand oder mit Hilfe von Analysetools durchgeführt werden. Meistens werden Analysetools zur Hilfe genutzt, da die Arbeit per Hand sehr monoton und deshalb auch fehleranfällig ist. Die Werkzeuge durchlaufen den Code wie ein Compiler und erstellen damit eine Analyse und einen Fehlerbericht für den untersuchten Quellcode. [1, S. 99 – 102]

Von diesen Analysetools gibt es eine große Bandbreite am Markt mit verschiedener Funktionalität und Anwendungsgebiet. In Kapitel 3 werden drei Beispieltools kurz vorgestellt. Bei der Nutzung von Analysetools bekommt der Anwender nach der Analyse einen Fehlerbericht mit gefundenen Fehlern. In diesem Fehlerbericht gibt es falsch positive und falsch negative Fehler. Falsch positive Fehler sind Fehler, die das Tool gefunden hat, obwohl der Code an dieser Stelle korrekt ist oder absichtlich so geschrieben wurde. Falsch negative Fehler sind Fehler, die das Analysetool nicht gefunden hat. Die Aufgabe des Entwicklers ist es, diesen Fehlerbericht durchzuarbeiten und zu entscheiden, bei welchen Meldungen der Code angepasst oder geändert werden muss und an welchen nicht. Das sollte für die spätere Entwicklung entsprechend markiert werden. So wird bei einer späteren Analyse nicht der gleiche Teil nochmal bearbeitet. Bei vielen Tools kann die Analysegenauigkeit eingestellt werden. Das führt zu vielen Meldungen mit einigen falsch positiven oder zu wenig Meldungen mit falsch negativen Fehlern. In der Praxis wird der Fehlerbericht von Entwicklern oft nicht komplett durchgearbeitet wenn dieser zu lang ist und Fehler bleiben im Code enthalten.

Auch der Compiler selbst führt bereits eine statische Codeanalyse durch. Ohne korrekte Syntax wird kein Programmcode kompiliert. Des Weiteren werden je nach Compiler bereits einige leicht zu findende Fehler bemerkt. Nicht deklarierte Variablen, die Über- oder Unterschreitung von Feldgrenzen bei statischer Adressierung oder nicht typgerechte Verwendung der Variablen bei strenggetypten Programmiersprachen werden beispielsweise

von Compiler gefunden. [1, S. 100 f.]

3 Anwendung der statischen Codeanalyse

Dieses Kapitel zeigt anhand eines kleinen Beispiels die Fehler, die ein Analysetool in einem Quellcode finden kann und gibt im Anschluss einen kurzen Überblick über verfügbare Analysetools auf dem Markt.

3.1 Analysebeispiel anhand eines fehlerhaften Quellcodes

Listing 1 zeigt ein fehlerhaftes C-Programm. Anhand dieses Beispiels soll gezeigt werden, welche Fehler ein Analysetool beispielsweise findet.

Listing 1: Gruenfelder2013)]Fehlerhaftes C-Programm als Analysebeispiel (Übernommen von [2, S. 56 f.])

```
#include <string.h>

char* MyName();
char cUartData = (char) 0xFA;

main()
{
    unsigned uiA = (unsigned) cUartData;
    int i, a[10] = {0,0,0,0,0,0,0,0,0,0};
    for (i = 0; i <= 10; i++)
{
a[i] += i;
    }
    for (i = 0; i<= 3; i++);
    {
        char szString[10];
        a[i] -= cUartData;
        strcpy(szString, MyName());
    }
}

char* MyName(int i)
{
    char name[11] = "Joe Jakeson";
    if (i) name[2] = 'i';
    return name;
}
```

Der erste Fehler wird in Zeile drei gefunden. In C ist für eine Funktion ohne Parameter definiert, dass in Klammern void angegeben werden muss. [2, S. 57 f.]

In Zeile acht kann die Typumwandlung zu Problemen führen. Von char auf unsigned wird zuerst impliziert von char in int umgewandelt. Wenn char vorzeichenbehaftet ist, wird das Vorzeichenbit von cUartData auf die neue Bitbreite erweitert. Die Variable uiA enthält dann nicht den Wert 0xFA, sondern den Wert 0xFFFFFFFA.[2, S. 57 f.]

Ein Analysetool merkt auch, dass in Zeile 9 eine Null bei der Initialisierung fehlt. In Zeile 12 wird deshalb die Indexgrenze der Variable a überschritten. Auch diesen Fehler meldet das Analysetool. Am Ende der for-Schleife in Zeile 14 findet das Analysetool den unabsichtlichen Strichpunkt. [2, S. 57 f.]

Da für die main-Methode kein Rückgabewert angegeben ist, nimmt der Compiler den Rückgabewert int an. Es gibt aber kein return Statement mit einem Rückgabewert int. [2, S. 57 f.]

Dieses kurze Beispiel zeigt bereits, dass ein Analysetool viele Fehler finden kann und die manuelle Fehlersuche eines Entwicklers deutlich erleichtert. Es zeigt aber auch, dass ein Entwickler bei vielen Meldungen entscheiden muss, welche Fehler behoben werden müssen und an welchen Stellen keine Quellcodeänderungen notwendig sind.

3.2 Wichtige Analysetechniken bei der statischen Codeanalyse

Bei der statischen Analyse von Software werden von den Analysetools verschiedene Methoden genutzt. Zwei der wichtigsten sind die Datenflussanalyse und die Kontrollflussanalyse.

3.2.1 Datenflussanalyse

Die Datenflussanalyse wird genutzt, um mögliche dynamische Fehler beim Programmablauf im statischen Zustand zu finden.
Für jede Variable gibt es drei Zustände:

- definiert (d): Die Variable erhält einen Wert zugewiesen
- referenziert (r): Der Wert der Variable wird ausgelesen
- undefiniert (u): Die Variable hat keinen definierten Wert

Im Zusammenhang mit der Datenflussanalyse wird oft von Datenflussanomalien gesprochen. Dabei geht um Situationen, die nicht zwingend zu einem Fehler führen, je nach Eingabe aber zu einem Fehler führen können. Ein Beispiel dafür ist eine du-Anomalie. Das ist eine Variable, die definiert wird, der Wert wird aber nie ausgelesen und verwendet. Das muss nicht zu einem Fehler führen, es ist aber fraglich, wieso dieser Wert für

diese Variable definiert wurde. In einfachen Beispielen sind diese Anomalien leicht zu sehen, in umfangreicheren Projekten können aber zwischen den betroffenen Zeilen viele weitere Codezeilen vorhanden sein und die Anomalie ist bei einem Review sehr schwer zu finden. Ein Analysetool findet diese Anomalien immer. [1, S. 102 f.]

3.2.2 Kontrollflussanalyse

Eine weitere Analysetechnik ist die Kontrollflussanalyse. Dabei wird aus dem Programmcode ein Kontrollflussgraph erstellt. Dieser Kontrollflussgraph zeigt den Ablauf des Programms. Anweisungen oder Sequenzen von Anweisungen werden dabei als Knoten dargestellt, IF-Abfragen oder Schleifen können dabei beispielweise zu anderen Anweisungen führen und verzweigen somit den Graphen. Durch die übersichtliche Darstellung in einem Graphen können Anomalien und komplexe und dadurch schnell fehleranfällige Strukturen leicht erkannt werden. [1, S. 104]
Im Zusammenhang mit der Kontrollflussanalyse geht es oft um die zyklomatische Zahl. Diese Zahl ist eine Metrik, um zu messen, wie komplex ein Programmteil ist.
Die zyklomatische Zahl wird über folgende Formel berechnet:
v(G) = e - n + 2
v(G) = zyklomatische Zahl des Graphen G
e = Anzahl der Kanten des Kontrollflussgraphen
n = Anzahl der Knoten des Kontrollflussgraphen

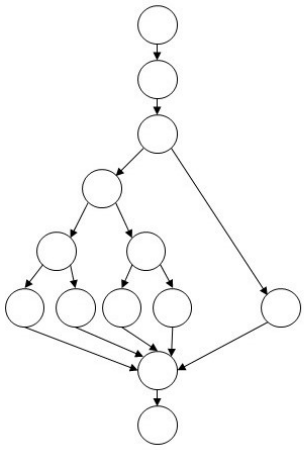


Abbildung 1: Kontrollflussgraph [1, S. 106]

Abbildung 1 zeigt ein Beispiel für einen Kontrollflussgraphen einer Funktion. In diesem Fall wäre die zyklomatische Zahl v(G) = 6. Je höher diese Zahl ist, desto komplexer ist schwerer wartbar ist der Programmcode. [1, S. 105 f.]

3.3 Beispieltools für die Unterstützung bei der statischen Codeanalyse

Auf dem Markt gibt es eine Vielzahl an Softwaretools, die zur Unterstützung bei der statischen Codeanalyse genutzt werden können. Die Auswahl des genutzten Tools hängt dabei sehr von der benötigten Funktionalität ab. Je nach Tool werden verschiedene Programmiersprachen unterstützt, es können verschiedene Fehler gefunden werden und auch die Anzeige der gefundenen Fehlern variiert dabei stark. So gibt es beispielsweise Tools, die über die Kommandozeile bedient werden, es gibt aber auch welche, die in die Entwicklungsumgebung mit integriert werden und dann die Fehler beispielsweise direkt in Visual Studio angezeigt werden. Wieder andere Tools nutzen für die Anzeige der Fehler eine eigene graphische Oberfläche, mit der die Fehler aufbereitet und angezeigt werden. Es gibt kostenlose Tools und auch solche, die nur mit einer kostenpflichtigen Lizenz genutzt werden können. Im Folgenden werden ein paar Beispiele kurz vorgestellt. Dabei stellt dies nicht die besten Tools oder eine Rangliste dar, sondern es ist lediglich eine Auswahl einiger sehr viel genutzter Tools.

FindBugs ist ein Open-Source-Tool für die statische Codeanalyse von Java-Code, das potenzielle Bugs und Sicherheitslücken identifiziert. Es bietet automatisierte Prüfungen für häufige Fehlermuster in Java-Programmen. Die gefundenen Fehler werden kategorisiert, um den Umgang mit den Fehlern zu erleichtern. FindBugs kann in bekannten Entwicklungsumgebungen wie beispielsweise Eclipse über ein Plugin installiert und genutzt werden. Über ein eigenes Konfigurationsfenster können dann verschiedene Einstellungen vorgenommen werden. Beispielsweise kann eingestellt werden, ob Fehler als Info, Warning oder Fehler gemeldet werden sollen. Dabei kann zusätzlich nach der Priorität der Fehler unterschieden werden. So können Fehler der schwerwiegensten Kategorie Scariest als Fehler und Fehler der Kategorie Of Concern als Info angezeigt werden. Es können auch nur Fehler aus bestimmten Bereichen wie Performance oder Security angezeigt oder ausgeblendet werden. [3]

Snyk ist ein kostenpflichiges Tool. Die Preise beginnen ab 57 Dollar pro Monat je Nutzer. Es gibt eine kostenlose Testversion die 100 Tests pro Monat zulässt. Snyk hat als große Stärke das Erkennen von Sicherheitsmängeln. Auch für Snyk gibt es verschiedene Plugins, die einfach in die genutzte Entwicklungsumgebung integriert werden können. Snyk nutzt künstliche Intelligenz für die Codeverbesserungsvorschläge. Im Gegensatz zu beispielsweise ChatGPT werden bei Snyk durch eine hybride Methodik keine Halluzinationen, also möglicherweise falsche Codeausschnitte, erstellt. Der Nutzer hat nach abgeschlossenem Scan immer die Möglichkeit, sich den neu generierten Code anzusehen bevor er entscheidet, ob die neuen Ausschnitte übernommen oder nochmal bearbeitet werden sollen. [4]

SonarQube ist gut dafür geeignet die Codequalität hoch zu halten. Eine Besonderheit ist die Analyse während dem Programmieren. So bekommt der Nutzer bereits während dem Coding eine Rückmeldung über mögliche Fehler. Die Kosten betragen nach einer

14 tägigen Testversion 150 Euro für jede Lizenz pro Jahr. Genutzt werden kann SonarQube für viele verschiedene Programmiersprachen. Es werden bei der Analyse über 5000 Programmierregeln beachtet. [5]

4 Herausforderungen der statischen Codeanalyse

Mit Hilfe der statischen Codeanalyse können viele Fehler in frühzeitigen Entwicklungsphasen gefunden werden, es können aber auch nicht alle Fehler aufgedeckt werden. Erkannt werden können keine logischen Fehler, die eigentliche Funktion der Software sowie das Zusammenspiel mit weiteren Software- oder Hardwarebausteinen. Dazu wird die dynamische Codeanalyse benötigt.

Der Aufwand für die statische Codeanalyse ist sehr hoch. Die meisten Softwaretools haben eine Einstellmöglichkeit, wie genau die Analyse erfolgen soll. Dabei werden dann entweder weniger Fehlermeldungen produziert, was zu mehr unentdeckten negativen Fehlern führt oder sehr viele Fehlermeldungen wodurch viele falsch positive Fehlermeldungen angezeigt werden. Die Aufgabe des Entwicklers ist es, nach der durchgeführte Analyse alle Fehlermeldungen durchzugehen und zu entscheiden, an welchen Stellen der Code angepasst werden muss und an welchen Stellen ein kurzer Kommentar reicht, dass diese Fehlermeldung so in Ordnung ist und nichts verändert werden muss. Diese Kommentare sind notwendig, damit bei späteren Weiterentwicklungen nicht an der gleichen Stelle wieder entschieden werden muss, ob eine Codeänderung nötig ist. Je nach Anzahl der Fehlermeldungen kann diese Analyse des Entwicklers sehr aufwendig sein und wird deshalb in der Praxis nicht immer in ausreichendem Umfang durchgeführt.

5 Zukunftsausblick und Fazit

5.1 Zukunftsausblick

Wie in anderen Bereichen wird auch bei der statischen Codeanalys versucht, die Automatisierung der Abläufe voranzutreiben und damit die Zeit der Entwickler für andere Tätigkeiten zu nutzen. Dabei wird auch die künstliche Intelligenz in diesem Bereich in Zukunft eine immer wichtigere Rolle spielen. Dadurch werden die Codeverbesserungsvorschläge der Analysetools weiter verbessert werden. Durch die zunehmende Bedrohung durch Cyberangriffe wird die Sicherheit von Software immer wichtiger. Die statische Codeanalyse ist eine Möglichkeit, um Sicherheitslücken frühzeitig zu erkennen und zu beheben.

5.2 Fazit

Die statische Codeanalyse ist in der Entwicklung von Software ein wichtiges Werkzeug, um Fehler in frühren Entwicklungsphasen zu finden. Zusammen mit der dynamischen

Codeanalyse kann somit die Qualität von Software erhöht werden. Auch wenn der Aufwand für das Testen von Software sehr groß ist, ist es trotzdem sinnvoll. Ein frühzeitig gefundener Fehler spart Zeit und Geld im weiteren Entwicklungsprozess. Das Einhalten von Programmierrichtlinien hilft bei der Zusammenarbeit im Team und bei späterer Wartung oder Erweiterung der Software. Mit guter statischer Codeanalyse wird von Anfang an die Einhaltung dieser Programmierrichtlinien sichergestellt.

Literatur

[1] A. Spillner. *Basiswissen Softwaretest*. 5.Auflage. dpunkt.verlag, 2012. ISBN: 978-3-86490-024-2.

[2] S. Grünfelder. *Software-Test für Embedded Systems*. 1.Auflage. dpunkt.verlag, 2013. ISBN: 978-3-86490-048-8.

[3] *Introduction to FindBugs*. 2024. URL: https://www.baeldung.com/intro-to-findbugs (besucht am 01. 06. 2024).

[4] *KI konzipiert speziell für sichere Entwicklung*. 2024. URL: https://snyk.io/de/platform/deepcode-ai/ (besucht am 08. 06. 2024).

[5] *the code quality for tool for better code*. 2024. URL: https://www.sonarsource.com/products/sonarqube/ (besucht am 08. 06. 2024).

Eidesstattliche Erklärung

Ich versichere, dass ich die vorliegende Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe, und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat.

Alle Ausführungen der Arbeit, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Bei der Erstellung dieses Dokuments wurde keinerlei auf künstliche Intelligenz (KI)-basierte Software verwendet.

Augsburg, 16. Juni 2024

()